January 4, 2013

The TUnfold package: user manual

Stefan Schmitt, DESY, Notkestraße 85, 22607 Hamburg

Email: Stefan.Schmitt@desy.de

Abstract

TUnfold is a package with provides functionality for correcting migration and background effects for multi-dimensional distributions. This document gives a user-oriented technical description of the package, valid for the version number 17.2.

1 Package overview

The TUnfold package provides algorithms to correct measured distributions for migration effects. The algorithm is based on least-square fitting and Tikhonov regularisation, it is described in [1]. In this document, details of the technical implementation and of the user interface are described. It is assumed that the reader is familiar with the algorithm [1].

The package is written in the C++ programming language. It consists of the four classes `TUnfold`, `TUnfoldSys`, `TUnfoldDensity` and `TUnfoldBinning`. The package is tied to the ROOT analysis framework [2].

1.1 Root versions and TUnfold versions

As of root version 5.22, some version of the TUnfold package is distributed together with the root software. Table 1 summarizes the connection between TUnfold version and distributed root versions. The most recent Root version 5.36 does not include the full functionality of TUnfold. However, it is possible to download the latest TUnfold version 17.2 and use it together with any ROOT relase, even if it comes along with an older version of TUnfold. In order to achieve this, in the distributed TUnfold 17.2 package, the classes

ROOT	TUnfold	Supported TUnfold classes
5.21 and earlier	–	–
5.22	V6	`TUnfold`
5.23-5.25	V13	`TUnfold TUnfoldSys`
5.27	V15	`TUnfold TUnfoldSys`
5.28-5.36	V16.0	`TUnfold TUnfoldSys`
6.00	V17.1	`TUnfold TUnfoldSys TUnfoldDensity TUnfoldBinning`
	V17.2	`TUnfold TUnfoldSys TUnfoldDensity TUnfoldBinning`

Table 1: correspondence of distributed ROOT versions and TUnfold versions.

have been renamed: the class `TUnfold` is named `TUnfoldV17`, the class `TUnfoldSys` is named `TUnfoldSysV17`, etc. In the header files, statements like

```
#define TUnfold TUnfoldV17
```

have been added, such that the renamed classes are accessible under their usual name.

1.2 TUnfold distribution

The TUnfold package is available for download here [3]. The package comes as a gzipped tar archive. The archive should contain the files given in table 2. TUnfold is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

README	notes on compiling
COPYING	licence file
tunfold_manual.tex	LaTex source of this manual
tunfold_manual_fig1.eps	Figure 1 of this manual
tunfold_manual_fig2.eps	Figure 2 of this manual
Makefile	default makefile for linux systems
altercodeversion.sh	auxillary script
TUnfold.h	header file providing the class TUnfoldV17
TUnfoldSys.h	header file providing the class TUnfoldSysV17
TUnfoldDensity.h	header file providing the class TUnfoldDensityV17
TUnfoldBinning.h	header file providing the class TUnfoldBinningV17
TUnfoldV17.cxx	implementation of the class TUnfoldV17
TUnfoldSysV17.cxx	implementation of the class TUnfoldSysV17
TUnfoldDensityV17.cxx	implementation of the class TUnfoldDensityV17
TUnfoldBinningV17.cxx	implementation of the class TUnfoldBinningV17
testUnfoldXX.C	example macros where XX=1, 2, 3, 4, 5a, 5b, 5c, 5d
docu.C	small program to test the documentation generated by ROOT.

Table 2: files distributed with the TUnfold package version 17.2.

TUnfold is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with TUnfold. If not, see `http://www.gnu.org/licenses/`.

1.3 Makefile

For many unix systems, the Makefile provided with this distribution is suitable for compiling the examples and the library. Note however, compilation only has been tested on selected systems. In general, modifications to the Makefile may be needed in order to compile the TUnfold package. The main commands from the Makefile are

make lib creates a shared library `libtunfold.so`.

make bin creates wrapper code to call the example macros and compiles them as standalone executables. For example the file `testunfold1.C` is created and compiled as executable `testunfold1`.

For using the TUnfold package, it is probably best to work through the example given by the four macros `testUnfold5a.C`, `testUnfold5b.C`, `testUnfold5c.C` and `testUnfold5d.C`.

1.4 Class overview

The four classes distributed with TUnfold are described briefly in the following. For most applications, the proper class to use is `TUnfoldDensity` and possibly also the class `TUnfoldBinning` to set up the analysis bins.

class TUnfold provides the core unfolding algorithm, matrix operations and methods to import from histograms or to export to histograms.

class TUnfoldSys adds functionality to the class `TUnfold` to treat background and systematic uncertainties.

class TUnfoldDensity adds functionality to the class `TUnfoldSys` to properly take into account bin widths and multi-dimensional distributions.

class TUnfoldBinning is used to tell the class TUnfoldDensity how the bins in complex binning schemes are arranged.

Table 3 gives a summary of the most important methods available with the TUnfold package.

2 Histograms and binning schemes

ROOT histograns are used to exchange information between the TUnfold package and the user. Internally, the algorithm works with vectors to store the bins of the input and output distributions. In the following, the relations of histogram bins to vector elements are discussed.

2.1 Use of bin maps with class TUnfold and TUnfoldSys

When importing data into the classes `TUnfold` or `TunfoldSys`, only the bin contents and bin errors of the histograms are relevant. The bin edges are not used. When extracting data into an existing histogram, the binning of that histogram is not checked. It is up to the user to book a histogram with the proper binning. It is then possible to change the mapping of the vector components to histogram bins. The mapping function is stored as an array of integer numbers and is denoted "bin map". Each element of the bin map corresponds to one of the bins in the unfolding result. The value stored in the bin map indicates the destination histogram bin in which the result shall be stored. It is possible to add up several bins of the unfolding result simply by using the same destination bin number for different elements of the bin map. The concept of the bin map is illustrated in figure 1.

Run the unfolding	
Method	**Description**
constructor	define matrix of migrations and basic regularisation scheme
`SetInput()`	define measurement
`AddSysError()`	set a systematic uncertianty
`SubtractBackground()`	set a background source
`DoUnfold()`	unfold once, with fixed tau
`ScanLcurve()`	scan L-curve (unfold multiple times) and determine tau
`ScanTau()`	scan correlations (unfold multiple times) and determine tau
Retreive unfolding results	
Method	**Description**
`GetOutput()`	unfolding result
`GetEmatrixTotal()`	total error matrix
`GetRhoItotal()`	total global corelations
`GetDeltaSysSource()`	systematic shifts from one systematic error
`GetDeltaSysBackgroundScale()`	systematic shifts from one background scale error
`GetEmatrixSysUncorr()`	error matrix from uncorrelated uncertainty on migration matrix
`GetEmatrixSysBackgroundUncorr`	error matrix from uncorrelated uncertainty on one background source
`GetEmatrixInput`	error matrix from input errors
Retreive unfolding error matrix (only when using class TUnfold)	
Method	**Description**
`GetEmatrix`	(deprecated) get error matrix
`GetRhoI`	(deprecated) get global corelations

Table 3: basic methods required to use the unfolding package. The table lists the name of the method and a short description.

2.2 Binning schemes and TUnfoldDensity

For the class `TUnfoldDensity` the bins are structured in a "binning scheme" using the class `TUnfoldBinning`. For one-dimensional unfolding problems, the binning schemes are constructed directly from the matrix of migrations. The user does not have to deal with the class `TUnfoldBinning`. For more complex problems, involving multi-dimensional distributions, multiple channels or unfolding of background normalisation factors, the corresponding binning schemes have to be defined by the user. The binning scheme information is used when setting up the regularisation scheme. In addition, it is used to create histograms having proper bin widths when extracting data from the class `TUnfoldDensity`. Furthermore, the binning scheme provides functionality to find the proper bin numbers when filling the histogram of migrations or the histogram of measurements.

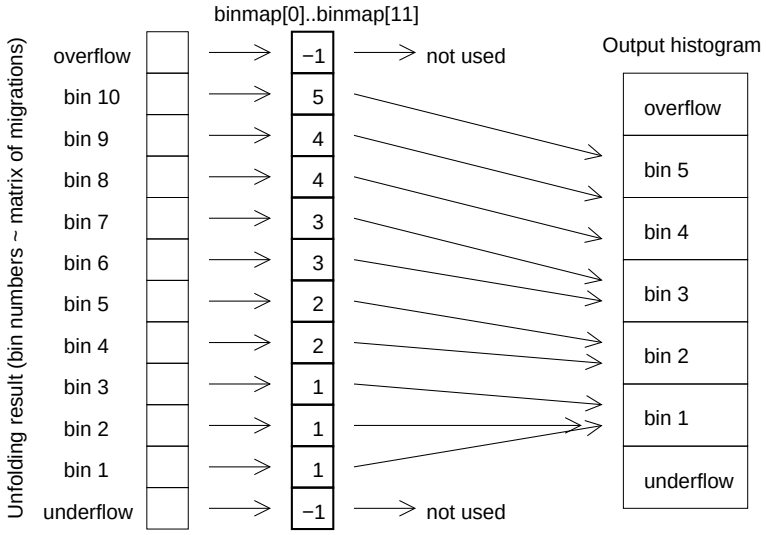

Figure 1: For the classes `TUnfold` and `TunfoldSys`, the bin map defines which bins of the unfolding result are stored in which histogram bin. In the example, 10 bins are mapped to 5 bins.

2.3 Unfolding one-dimensional distributions in TUnfoldDensity

When unfolding one-dimensional distributions, it is most convenient to book and fill the histogram of migrations using the bins as required for the analysis. There is no need to define binning schemes. The matrix of migrations is stored as a two-dimensional histogram, where on one axis the truth bins are arranged. It is possible to have underflow and overflow bins for the truth parameters. If these are present, their content is also unfolded from the data. On the other axis there is the reconstructed quantity, again with the appropriate bins. Here, it is not possible to use the underflow and overflow bins for measurements. Instead, these bins are used to count events which originate from a specific truth bin but where the reconstructed quantity is not available. An example is given in figure 2.

2.4 Complex binning schemes

The class `TUnfoldBinning` provides means to map bins originating from one or several multi-dimensional distributions on a single histogram axis and back. The multi-dimensional distributions are arranged in a tree structure.

For the truth parameters, the branches of the tree structure could correspond to different decay channels, signal and background, etc. Each branch then holds several bins, in most cases in the form of a multi-dimensional histogram. Similarly, for the reconstructed parameters, the branches of the tree structure could correspond to different reconstructed channels and various control distributions. So in general, there are two "binning trees", a tree of truth bins and a tree of reconstructed bins.

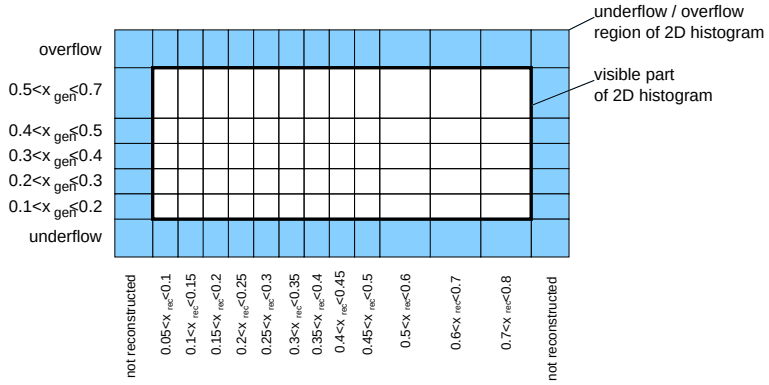

Figure 2: The matrix of migrations in the case of one-dimensional unfolding is illustrated. The truth parameter x_{gen} has five non-equidistant bins, ranging from 0.1 to 0.7 plus underflow and overflow bins (seven bins in total). The reconstructed parameter x_{rec} has twelve bins ranging from 0.05 to 0.8. The underflow and overflow bins in x_{rec} are used to count the non-reconstructed events.

When filling the histogram of migrations, the proper bin numbers both in the tree of truth bins and in the tree of reconstructed bins have to be determined. The bin number i_{gen} on truth level is determined as follows: first, the appropriate branch is determined, for example by deciding on the event type (signal or background). The method `FindNode()` may be used to locate a branch in the tree using its name. Next, using the truth parameters, the bin number i_{gen} is calculated using the method `GetGlobalBinNumber()` on the branch. Below this is illustrated in a code fragment, assuming the signal branch contains a three-dimensional histogram with the variables `xTrue`, `yTrue`, `zTrue`.

```
Int_t iGen;
const TUnfoldBinning *signalBranch=generatorBinning->FindNode("signal");
iGen=signalBranch->GetGlobalBinNumber(xTrue,yTrue,zTrue);
```

The bin number i_{rec} in the tree of reconstructed bins is determined in a similar manner, for example by deciding on the reconstructed channel and then using the appropriate reconstructed quantities to calculate the bin number. If the event was not reconstructed, the special bin number $i_{\text{rec}} = 0$ must be used.

Finally, the event weight w_{gen} is filled in the corresponding bin of the two-dimensional histogram of migrations. Sometimes there is a secondary event weight w_{rec} to account for detector efficiency corrections. In order to get the proper efficiency correction from the unfolding, the event must be filled twice into the histogram of migrations: first, the histogram of migration is filled at the position $(i_{\text{gen}}, i_{\text{rec}})$ using the weight $w_{\text{gen}} \times w_{\text{rec}}$. Next, the histogram of migration is filled again, this time at the position $(i_{\text{gen}}, 0)$ using the event weight $w_{\text{gen}} \times (1 - w_{\text{rec}})$.

For data, the procedure to determine the bin number i_{rec} is applied for the reconstructed quantities only, and a one-dimensional histogram is filled.

Setting up binning schemes with `TUnfoldBinning` is illustrated in the example macro `testUnfold5b.C`. How to use the binning scheme to fill histograms is illustrated in `testUnfold5c.C`. Unfolding and extracting distributions using the binning scheme is illustrated in `testUnfold5d.C`.

2.5 XML interface to binning schemes

An XML interface is provided for the binning schemes. The DTD definition is repeated here. There is a method `TUnfoldBinning::WriteDTD()` which saves the DTD to a file.

```
<!ELEMENT TUnfoldBinning (BinningNode)+ >
<!ELEMENT BinningNode (BinningNode+|Axis|Bins) >
<!ATTLIST BinningNode name ID #REQUIRED firstbin CDATA "-1"
    factor CDATA "1.">
<!ELEMENT Axis (Axis|Bin)+ >
<!ATTLIST Axis name ID #REQUIRED lowEdge CDATA #REQUIRED>
<!ELEMENT Bin EMPTY>
<!ATTLIST Bin width CDATA #REQUIRED location CDATA #IMPLIED
    center CDATA #IMPLIED>
<!ELEMENT Bins (BinLabel)* >
<!ATTLIST Bins nbin CDATA #REQUIRED>
<!ELEMENT BinLabel EMPTY>
<!ATTLIST BinLabel index CDATA #REQUIRED name CDATA #REQUIRED>
```

There are methods `ExportXML` and `ImportXML` to write or read binning trees in XML format. One XML file may contain several binning schemes.

It is probably best to study the example macros `testUnfold5a.C`–`testUnfold5d.C` to find out how the XML interface works. In the example `testUnfold5a.C`, pseudo events are written to root files "testUnfold5_data.root", "testUnfold5_signal.root", "testUnfold5_background.root". In `testUnfold5b.C`, binning schemes are set up and then stored as XML in a file named "testUnfold5binning.xml". In `testUnfold5c.C` the XML file is read and the binning schemes are used when looping over the data, signal and background events. Histograms required for the unfolding are filled. The histograms and the binning schemes are then stored in another root file "testUnfold5_histograms.root". Finally, the macro `testUnfold5d.C` reads back that root file and runs the unfolding. One could try to edit the XML file and run `testUnfold5c.C` and `testUnfold5d.C` repeatedly to see the effects.

3 Regularisation

For unfolding, regularisation conditions are imposed. The regularisation is given by the scalar product $(\tau Lz, \tau Lz)$, where z is the difference of the unfolding result to a bias

vector and L is a matrix describing the regularisation scheme. The parameter τ gives the strength of the regularisation. The number of columns of L is identical to the number of unfolded bins. The number of rows reflects the number of regularisation conditions, it may be different from the number of columns.

3.1 Basic regularisation types

Three basic types of regularisation are supported: `kRegModeSize`, `kRegModeDerivative`, `kRegModeCurvature`. The type of regularisation may be specified with the constructor of either of the classes `TUnfold`, `TUnfoldSys`, `TUnfoldDensity` as the third argument. In that case, the given basic regularisation is applied to all bins.

The simplest regularisation condition is given by `kRegModeSize`, corresponding to the case where L is the unity matrix. The matrix L is diagonal and does not mix different bins. The regularisation is given by $\tau^2 \sum z_i^2$.

For the condition `kRegModeDerivative`, the matrix L calculates differences $x_j - x_i$, thus approximating first derivatives. In that case, the structure of the input bins matters, because differences should be calculated between adjacent bins only. For one-dimensional distributions this done by simply setting $j = i + 1$. For two-dimensional distributions, derivatives may be defined along both dimensions and the relation is getting more complicated. When using the classes `TUnfoldDensity` and `TUnfoldBinning`, the relation of the bins is known and appropriate regularisation schemes are defined automatically.

For the condition `kRegModeCurvature`, the matrix L approximates second derivatives $(x_k - x_j) - (x_j - x_i)$. Similar to the case of `kRegModeDerivative` the corresponding matrix structure may get rather complicated in case of multi-dimensional distributions, and is most conveniently handled through the use of the classes `TUnfoldDensity` and `TUnfoldBinning`.

3.2 Non-standard regularisation schemes

Sometimes it is useful to set up non-standard regularisation schemes. When using the class `TUnfoldDensity` with user-defined binning schemes, there is additional control over the regularisation scheme. One may select modifications of the calculation of L such that the components of z are normalized to the corresponding bin widths prior to calculating the regularisation conditions. Furthermore, it is possible to take into account the bin widths for the calculation of the first or second derivatives. One may also set specific normalisation factors or normalisation functions with the binning scheme and use those to modify the normalisation of z in the calculation of the regularisation.

For binning schemes based on trees with several branches it is possible to restrict the regularisation to one of the branches or to set up dedicated regularisation schemes for each of the branches (method `RegularizeDistribution()`). For multi-dimensional distributions it is possible to exclude underflow or overflow bins or to exclude derivatives calculated along specific axes from the regularisation. Ultimatelty, it is also possible to define arbitrary regularisation conditions by adding single rows to the matrix L (method `AddRegularisationCondition()`).

4 Determination of τ

One of the frequent questions related to the regularized unfolding method implemented in the TUnfold package is the choice of the regularisation parameter τ. If τ is too small, there is no regularisation. If τ is too large, the unfolding result is biased strongly by the regularisation condition. In the TUnfold package, two basic methods to determine the regularisation have been implemented, the L-curve scan and the minimisation of correlations.

4.1 L-curve scan

The L-Curve scan is available with the classes `TUnfold`, `TUnfoldSys` and `TUnfoldDensity`. The method is named `ScanLcurve`. It works as follows: the unfolding is repeated for a number of points with different τ, for example $n_p = 30$. A parametric curve of two variables $X(\tau)$ and $Y(\tau)$ is calculated. The exact definition of these variables is given in [1]. The optimal chioce of τ is determined as the position having the largest curvature ("kink") in the (X, Y) plane. For scanning the L-curve, the following parameters may be set: number of points n_p, minimum ($\tau_{\rm min}$) and maximum ($\tau_{\rm max}$) value of τ to scan. If $\tau_{\rm min} = \tau_{\rm max}$, the interval is chosen automatically. When runnung the scan, the following three curves are produced: $X(\tau)$, $Y(\tau)$ and $Y(X)$.

The scan proceeds as follows: Given a τ interval to scan, first the unfolding is performed for $\tau = \tau_{\rm min}$ and $\tau = \tau_{\rm max}$. Intermediate points are then inserted such that a most uniform population along the curve $X(\tau), Y(\tau)$ is achieved. Given two or more points (X_i, Y_i), ordered in the corresponding τ_i, a new point is inserted into the interval which has the largest size $S^2 = (X_{i+1} - X_i)^2 + (Y_{i+1} - Y_i)^2$ until $n_p - 1$ points have been calculated. The last point of the scan is inserted at the best choice of tau, determined from the set of $n_p - 1$ points.

4.2 Minimisation of correlation coefficients or other quantities

With the class `TUnfoldDensity` another method of determining τ is implemented. The method `ScanTau()` repeats the unfolding n_p times for different choices of τ. During that scan, the minimum of a function $Z(\tau)$ is determined. The possible choices of the function Z are summarized in table 4. They all depend on the calculation of global correlation coefficients ρ_i, which is described in [1]. When using the method `ScanTau()`, the following parameters have to be set: the number of points n_p, the minimum and maximum value of τ to scan and the mode (table 4). If the minimum and maximum value of τ agree, the scan range is determined automatically. In addition one may change the way the correlation coefficients ρ_i are calculated. The calculation may be restricted to one branch in the binning tree or may use all branches. Within the distributions it is possible to exclude underflow and overflow bins or to integrate over bins. The scan returns four curves: the curve $Z(\tau)$ and in addition the three curves also returned by `ScanLcurve()`. For a given interval in τ, $n_p - 1$ points are inserted such that large τ intervals are split into

Mode	definition of Z
`kEScanTauRhoAvg`	$Z = \frac{1}{n_{\rm bin}} \sum_i \rho_i$ (average global correlation)
`kEScanTauRhoMax`	$Z = \max_i \rho_i$ (maximum global correlation)
`kEScanTauRhoAvgSys`	$Z = \frac{1}{n_{\rm bin}} \sum_i \rho_{i,\rm sys}$ (average global correlation, including systematic errors)
`kEScanTauRhoMaxSys`	$Z = \max_i \rho_{i,\rm sys}$ (maximum global correlation, including systematic errors)
`kEScanTauRhoSquareAvg`	$Z = \frac{1}{n_{\rm bin}} \sum_i (\rho_i)^2$ (average of squares of global correlation coefficients)
`kEScanTauRhoSquareAvgSys`	$Z = \frac{1}{n_{\rm bin}} \sum_i (\rho_{i,\rm sys})^2$ (average of squares of global correlation coefficients including systematic errors)

Table 4: Choices of the function Z for implemented with the method `ScanTau()`.

two. Finally, using the set of $n_p - 1$ points, the position of the minimum is determined and the unfolding is repeated at the position of the minimum.

The scan of correlation coefficients has the desired property that correlations in the result are minimized. Ideally, the correlation coefficients are small and can be neglected. However, this has to be checked carefully.

A drawback of the method is that it often fails. In particular, this method can not be used with the `kRegModeSize` regularisation condition. For the regularisation methods `kRegModeDerivative` and `kRegModeCurvature`, the method is expected to work more reliably.

References

[1] S. Schmitt, JINST **7** (2012) T10003 [arXiv:1205.6201].

[2] R. Brun and F. Rademakers, Nucl. Instrum. Meth. A **389** (1997) 81.

[3] S. Schmitt, TUnfold version 17.2, `http://www.desy.de/~sschmitt/tunfold.html`.